C++/Java Performance Comparison for Distributed ITS Control Systems

Contract DBM-9713-NMS
TSR # 9901961
Document # M361-AR-002R0

March 30, 1999
By

Computer Sciences Corporation and PB Farradyne Inc

Table of Contents

1 Introduction

The purpose of this document is to compare the performance of the Java and C++ languages as they pertain to the development of an ITS control system. To that end, the tests included in this comparison have been developed to investigate the performance characteristics of those language features that are most frequently utilized in the creation of an ITS control system. The results of each test will be presented and described in detail, along with a statement regarding the significance of the test and a detailed discussion of why the test was considered important enough to include in this comparison.

2 Environment

The environment for these tests was as follows:
Dell Dimension XPS R400 PC
400 MHz Pentium II processor
192 MB RAM
Microsoft Visual C++
Sun JDK 1.2

All of the C++ software was built using full speed optimization. This build configuration is typically used when we build a system for field deployment. The Java code was run using the Just-In-Time (JIT) compiler that is shipped with the Java Runtime Environment.

3 Tests

3.1 Integer Math Test

Most of the mathematical operations performed in a software system are performed on integer values. Such operations include calculating index offsets, window coordinates, counter variables and time based calculations. Thus, it would be easy to rule out a language that performed poorly on calculations of this nature. The test was designed to test the array indexing and integer math capabilities inherent in the language. An array of 100000 integers was created. The test then copied each element of the array into a local integer variable, incremented it by one and copied the integer back into the array. Next, the test took another pass through the array, this time copying each element to a local variable, decreasing the value of the local variable by one and then copying it back into the array. On the third pass through the array, the test copied the value from the array to the local variable, multiplied it by two, and copied it back into the array. On the fourth and final pass through the array, the test copied the value to a local variable, divided it by two, then placed it back into the array. This test performed 400000 mathematical operations on array elements, 400000 array index operations, and 800000 integer copies. The entire test was run 500 times and the following results represent the average time required to perform the test.

C++ - 8 Milliseconds
Java - 10 Milliseconds

3.2 Floating Point Math Test

Although floating point math is not nearly as pervasive in an ITS control system as integer math, it is heavily utilized in some key components; namely the system map and incident detection algorithms. The system map uses floating-point variables to store the geographical locations of objects and bounding rectangles of maps. Calculations are performed to determine if a particular object is within a particular bounding rectangle, distance between points, etc. The incident detection algorithm performs floating-point math to “smooth" new detector data in with historical data for a link each time new detector data is received. The test was designed to test the array indexing and floating-point math capabilities inherent in the language. An array of 100000 doubles was created. The test then copied each element of the array into a local double variable, multiplied it by itself and copied the resulting double back into the array. Next the test took another pass through the array, this time copying each element to a local variable, dividing the value by 3.14 and then copying it back into the array. On the third pass through the array, the test copied the value from the array to the local variable, added it to itself, and copied it back into the array. On the fourth and final pass through the array the test copied the value to a local variable, subtracted 3.14, and then placed it back into the array. All tolled, this test performed 400000 floating-point mathematical operations on double values, 400000 array index operations, and 800000 double variable copies. The entire test was run 500 times and the following results represent the average time required to perform the test.

C++ - 17 Milliseconds
Java - 30 Milliseconds

3.3 String Manipulation Test

For the CHART II system the speed with which string manipulations can be performed is important because each CORBA object is referenced using a string identifier known as the Interoperable Object Reference (IOR). Thus, many comparisons must be done on string values

to determine if two object instances represent the same remote object. Additionally, it is common to run string-parsing algorithms on DMS and HAR messages. The String Manipulation Test was designed to compare the performance of string comparisons and string alterations between the Java supplied String object and the MFC CString object. The MFC CString object was chosen because it is highly probable this is the string variant that would be used for this project. Additionally, the GUI would definitely be implemented using MFC and, thus, the CString class. Each iteration of the test performs the following processing 10000 times: initialize two string objects to the constant string "Initial string for manipulation test", compare the string objects to each other, print an error message if they do not match. Next, convert one of the strings to all upper case characters, compare the strings to each other again, print an error message if they do match. The test was repeated 500 times and the following results represent the average time to complete the test.

C++ - 60 Milliseconds
Java - 76 Milliseconds

3.4 Object Method Call Test

This test was designed to test the efficiency with which the language passes objects as parameters and invokes statically bound methods. This test is important because an object oriented software system is composed of objects and these objects are passed to functions as parameters which, in turn, may invoke methods on the object in order to get it to perform a task or alter it's current state. Those methods that alter the state of an object are commonly referred to as mutator functions and those objects that define mutator functions are commonly referred to as mutable objects. This test, then, creates a mutable object, passes it to a function as a parameter and (the function) invokes a mutator method on that object. The object is passed by reference in C++. In Java, it is not possible to pass an object any other way. Additionally, the C++ mutator method was declared inline in order to fully optimize the call. This entire process is repeated 500000 times to comprise one running of the test. The test was run 500 times and the results that follow represent the average time required to complete the test.

C++ - 18 Milliseconds
Java - 7 Milliseconds

3.5 Polymorphic Method Call Test

This test was designed to test the efficiency with which the language handles the invocation of methods that are dynamically bound. A dynamically bound method is one whose actual implementing class is not known at compile time. Polymorphism is fundamental to the design and implementation of reusable object oriented software and is the cornerstone upon which the CHART II user interface prototype was built and upon which the entire CHART II system will be built. Polymorphism allows the map component to treat all objects uniformly. The map may simply tell each object to Render() itself and it is up to the DMS class to Render a DMS or the HAR class to render a HAR object. To the map it does not matter and this allows us to later add AVL objects which support the Render() polymorphic method without altering the map code. This test creates a new object of a derived type and stores it in a base class pointer. It then invokes a polymorphic method of that object 500000 times. The test was performed 500 times and the results presented below represent the average time to complete the test.

C++ - 10 Milliseconds
Java - 8 Milliseconds

3.6 Dynamic Type Test

Dynamic type testing is a particularly useful tool in the implementation and design of object oriented software systems. It allows the program to test a heterogeneous collection of base class objects and find if any of them are of a particular type. A typical use of dynamic type checking would be to test a collection of device objects to see which of them are actually capable of being used in an incident response scenario (which of them supports a particular interface). Although this type of checking can be approximated through type tags or collections of polymorphic methods each of these methods of performing dynamic type testing has significant drawbacks associated with it. This test creates an instance of a derived class object and stores it in a base class pointer. The base class pointer is tested 500000 times to see if it is actually pointing to an object which is an instance of the derived class (it is). If it is an instance, the pointer is cast to a derived class object and a method that is defined only in the derived class is invoked. If it is not an error message is printed. In C++ dynamic type testing is accomplished via the dynamic_cast<> operator and requires that your code be compiled with the RTTI (Run Time Type Identification) compiler flag enabled. Java supplies the instanceof operator for this functionality. The test was run 500 times and the results presented are the average time to complete the test.

C++ - 224 Milliseconds
Java - 8 Milliseconds

4 Conclusion

The results of these performance tests indicate that C++ is slightly more efficient than Java at low-level computational tasks but that Java makes up for this by providing superior support for object oriented design constructs. It is the opinion of the development team that either language would perform in an adequate manner to implement the CHART II system. We base this conclusion on the fact that the system is distributed in nature and will, therefore, spend far more time waiting for data transmission to and from remote system components than performing intense calculations.

Acronyms

API	Application Programming Interface
AVL	Automated Vehicle Location
AWT	Advanced Windowing Toolkit
CHART	Congested Highways Action Response Team
CORBA	Common Object Request Broker Architecture
DMS	Dynamic Message Sign
HAR	Highway Advisory Radio
GUI	Graphical User Interface
IOR	Interoperable Object Reference
ITS	Intelligent Transportation System
JDK	Java Development Kit
JFC	Java Foundation Class
JIT	Just in Time
JNI	Java Native Interface
MFC	Microsoft Foundation Class
ORB	Object Request Broker
RFP	Request for Proposal
RTTI	Run Time Type Identification

Appendix A - Test Output

A.1 Java Test Output

```
Java Language Performance Test Results
**************************************

Performing int array test for 100000 elements.

This test will create an array of the specified number of elements
It will then iterate through all elements in the array four times.
The first time it will copy an integer out of the array, increment it
and copy it back into the array.  The second pass will copy the element
out of the array, decrement it by one and copy it back into the array.
The third pass will copy the element out of the array, multiply it
and place it back in the array.  The final pass will copy each element
divide it by two, and place it back in the array.

Average time for int array test is 10 milliseconds for 500 repetitions.


Performing floating point test for 100000 elements.
This test will create an array of the specified number of elements
it will then iterate through all elements in the array four times.
The first time it will copy a double out of the array, multiply it
by itself and copy it back into the array.  The second pass will copy
the element out of the array, divide it by 3.14 then copy it back into the
array.
The third pass will copy the element out of the array, add it to itself
and place it back in the array.  The final pass will copy each element
subtract 3.14 from it and place it back in the array.

Average time for floating point test is 30 milliseconds for 500 repetitions.


Performing string manipulation test for 10000 strings.
This test will copy a string into a temporary string.  It will then
compare the two strings using a string sensitive comparison.  Next it
will convert the temporary string to upper case.  It will then compare
strings again.

Average time for string manipulation test is 76 milliseconds for 500
repetitions.


Performing mutable object test for 500000 method calls.
This test will create an object which has a single integer member
variable and a public method which, when called, increments this
member.  The class of this object will also have a static method
which takes a mutable object as its lone parameter.  This allows us
to test the efficiency with which the language passes parameters by
reference.  The test creates a mutable object and passes it to the
class method which then invokes the mutator method on the passed object.
```

```
Average time for mutable object test is 7 milliseconds for 500 repetitions.


Performing polymorphic test for 500000 method calls.
This test is designed to test the efficiency with which the language
handles method invocations which require dynamic binding.  The test
creates a derived class object and assigns it to a base class pointer.
It then invokes a virtual method on the base class object which will
be handled by the overriding derived class method.  Dynamic binding is
crucial to engineering reusable object oriented components.

Average time for polymporphic test is 8 milliseconds for 500 repetitions.


Performing dynamic type test for 500000 objects.
This test is designed to determine how efficiently the language allows
a program to determine if an object is an instance of a particular class
at run-time.  This type of testing is done in java via the instanceof
operator which is a built-in language feature.  The test will create an
instance of a derived class object and store it in a base class pointer.
It will then perform the dynamic cast to test if the object is a derived
class object.  If it is, the test will call a method which is defined in the
derived class only.  If it is not an error message will be printed

Average time for dynamic type test is 8 milliseconds for 500 repetitions.
```

A.2 C++ Test Output

```
C++ Language Performance Test Results
*************************************

Performing int array test for 100000 elements.

This test will create an array of the specified number of elements
it will then iterate through all elements in the array four times.
The first time it will copy an integer out of the array, increment it
and copy it back into the array.  The second pass will copy the element
out of the array, decrement it by one and copy it back into the array.
The third pass will copy the element out of the array, multiply it
and place it back in the array.  The final pass will copy each element
divide it by two, and place it back in the array.

Average time for int array test is 8 milliseconds for 500 repetitions.


Performing floating point test for 100000 elements.

This test will create an array of the specified number of elements
it will then iterate through all elements in the array four times.
The first time it will copy a double out of the array, multiply it
by itself and copy it back into the array.  The second pass will copy
```

the element out of the array, divide it by 3.14 then copy it back into the array.
The third pass will copy the element out of the array, add it to itself and place it back in the array. The final pass will copy each element subtract 3.14 from it and place it back in the array.

Average time for floating point test is 17 milliseconds for 500 repetitions.

Performing string manipulation test for 10000 strings.

This test will copy a string into a temporary string. It will then compare the two strings using a string sensitive comparison. Next it will convert the temporary string to upper case. It will then compare strings again.

Average time for string manipulation test is 60 milliseconds for 500 repetitions.

Performing mutable object test for 500000 method calls.

This test will create an object which has a single integer member variable and a public method which, when called, increments this member. The class of this object will also have a static method which takes a mutable object as its lone parameter. This allows us to test the efficiency with which the language passes parameters by reference. The test creates a mutable object and passes it to the class method which then invokes the mutator method on the passed object.

Average time for mutable object test is 18 milliseconds for 500 repetitions.

Performing polymorphic test for 500000 method calls.

This test is designed to test the efficiency with which the language handles method invocations which require dynamic binding. The test creates a derived class object and assigns it to a base class pointer. It then invokes a virtual method on the base class object which will be handled by the overriding derived class method. Dynamic binding is crucial to engineering reusable object oriented components.

Average time for polymorphic call test is 10 milliseconds for 500 repetitions.

Performing dynamic type test for 500000 objects.

This test is designed to determine how efficiently the language allows a program to determine if an object is an instance of a particular class at run-time. This type of testing is done in C++ via the dynamic_cast<> operator which requires that the code be built with RTTI enabled. The test will create an instance of a derived class object and store it in a base class pointer. It will then perform the dynamic cast to test if the object is a derived class object. If it is, the test will call a method which is defined in the derived class only. If it is not an error message will be printed.

```
Average time for dynamic type test is 224 milliseconds for 500000 objects.
```

Appendix B - C++ Source Code

B.1 SPEED.CPP

```
#include "afx.h"
#include "time.h"
#include <iostream.h>
#include "objects.h"
#include "stdlib.h"

// method declarations

void doIntArrayTest();
unsigned int intArrayTest(unsigned int len);

void doMutableObjectTest();
unsigned int mutableObjectTest(unsigned int reps);

void doPolymorphicTest();
unsigned int polymorphicTest(unsigned int reps);

void doDynamicTypeTest();
unsigned int dynamicTypeTest(unsigned int reps);

void doFloatingPointTest();
unsigned int floatingPointTest(unsigned int reps);

void doStringTest();
unsigned int stringTest(unsigned int reps);

unsigned int clocks_to_millis(clock_t start_clock, clock_t end_clock);


//main method
int main(int argc, char* argv[], char *envp[])
{
   cout << "C++ Language Performance Test Results" << endl;
   cout << "*************************************" << endl << endl;

   doIntArrayTest();
   doFloatingPointTest();
   doStringTest();
   doMutableObjectTest();
   doPolymorphicTest();
   doDynamicTypeTest();

   return 1;
}


//driver for integer array test
void doIntArrayTest()
{
   int            reps = 500;
```

```
   unsigned int   elems = 100000;
   unsigned int   total_time = 0;

   cout << "Performing int array test for "<< elems << " elements." << endl
<< endl;
   cout << "This test will create an array of the specified number of
elements " << endl;
   cout << "it will then iterate through all elements in the array four
times." << endl;
   cout << "The first time it will copy an integer out of the array,
increment it" << endl;
   cout << "and copy it back into the array.  The second pass will copy the
element" << endl;
   cout << "out of the array, decrement it by one and copy it back into the
array." << endl;
   cout << "The third pass will copy the element out of the array, multiply
it " << endl;
   cout << "and place it back in the array.  The final pass will copy each
element" << endl;
   cout << "divide it by two, and place it back in the array." << endl <<
endl;

   for(int x = 0; x < reps; x++)
   {
      total_time += intArrayTest(elems);
   }
   cout << "Average time for int array test is "<< total_time/reps << "
milliseconds for " << reps << " repetitions." << endl << endl << endl;
}


/****************************************************************
   This test is designed to test the speed of array manipulation
   and basic mathematical functions for basic types.  While clocking
   it will create an array of the specified number of ints and walk
   it four times.  Each time it will pull out an element and perform
   a basic mathematical operation, then it will replace the element
   in the array.
****************************************************************/
unsigned int intArrayTest(unsigned int len)
{
      int test = 0;
      unsigned int x = 0;

      clock_t start_clock = clock();

      int *myints = new int[len];

      for(x = 0; x < len; x++)
      {
         test = myints[x];
         test++;
         myints[x] = test;
      }
      for(x = 0; x < len; x++)
      {
         test = myints[x];
```

```
         test--;
         myints[x] = test;
      }
      for(x = 0; x < len; x++)
      {
         test = myints[x];
         test*=2;
         myints[x] = test;
      }
      for(x = 0; x < len; x++)
      {
         test = myints[x];
         test/=2;
         myints[x] = test;
      }

      delete[] myints;
      clock_t end_clock = clock();

      unsigned int millis = clocks_to_millis(start_clock, end_clock);

      return millis;
}


//driver for mutable object test
void doMutableObjectTest()
{
   int            reps = 500;
   unsigned int   calls = 500000;
   unsigned int   total_time = 0;

   cout << "Performing mutable object test for "<< calls << " method calls."
<< endl << endl;
   cout << "This test will create an object which has a single integer
member" << endl;
   cout << "variable and a public method which, when called, increments this"
<< endl;
   cout << "member.  The class of this object will also have a static method"
<< endl;
   cout << "which takes a mutable object as its lone parameter.  This allows
us" << endl;
   cout << "to test the efficiency with which the language passes parameters
by" << endl;
   cout << "reference.  The test creates a mutable object and passes it to
the" << endl;
   cout << "class method which then invokes the mutator method on the passed
object." << endl << endl;

   for(int x = 0; x < reps; x++)
   {
      total_time += mutableObjectTest(calls);
   }
   cout << "Average time for mutable object test is "<< total_time/reps << "
milliseconds for " << reps << " repetitions." << endl << endl << endl;
}
```

```
/*****************************************************************
   This test is designed to test static method calls and simple
   object state change calls.  While clocking it will create a
   new and call a static method which will call a mutator
   method on the object for the specified repititions.
*****************************************************************/
unsigned int mutableObjectTest(unsigned int reps)
{
   clock_t start_clock = clock();

   MutableObj mo;
   for(unsigned int x = 0; x < reps; x++)
   {
      MutableObj::MutateObject(mo);
   }

   clock_t end_clock = clock();

   return clocks_to_millis(start_clock, end_clock);
}


//Simple function to compute milliseconds from clock ticks...
unsigned int clocks_to_millis(clock_t start_clock, clock_t end_clock)
{
   clock_t ticks = end_clock - start_clock;
   double millisd = ((double)ticks/(double)CLOCKS_PER_SEC)*(double)1000;
   return millisd;
}


//driver for the polymorphic call test
void doPolymorphicTest()
{
   int            reps = 500;
   unsigned int   calls = 500000;
   unsigned int   total_time = 0;

   cout << "Performing polymorphic test for "<< calls << " method calls." <<
endl << endl;
   cout << "This test is designed to test the efficiency with which the
language" << endl;
   cout << "handles method invocations which require dynamic binding.  The
test" << endl;
   cout << "creates a derived class object and assigns it to a base class
pointer." << endl;
   cout << "It then invokes a virtual method on the base class object which
will " << endl;
   cout << "be handled by the overriding derived class method.  Dynamic
binding is" << endl;
   cout << "crucial to engineering reusable object oriented components." <<
endl << endl;

   for(int x = 0; x < reps; x++)
   {
      total_time += polymorphicTest(calls);
```

```
   }
   cout << "Average time for polymorphic call test is "<< total_time/reps <<
" milliseconds for " << reps << " repetitions." << endl << endl << endl;
}


/*****************************************************************
   The purpose of this test is to determine the efficiency with which
   the language processes method calls which require dynamic binding.
*****************************************************************/
unsigned int polymorphicTest(unsigned int reps)
{
   clock_t start_clock = clock();

   BaseObject *pmo = new DerivedObject();
   for(unsigned int x = 0; x < reps; x++)
   {
      pmo->polymorphicCall();
   }
   delete pmo;

   clock_t end_clock = clock();

   return clocks_to_millis(start_clock, end_clock);
}


//base class polymorphic call from poly call test... just prints an error
void BaseObject::polymorphicCall()
{
   //this should never happen
   cout << "Error call went to base class." << endl;
}


//override of polymorphic call in polymorphic call test
void DerivedObject::polymorphicCall()
{
   m_counter++;
}


//method which is declared only in the derived object... used in dynamic type
test
void DerivedObject::derivedOnlyCall() inline
{
   m_counter++;
}


//driver for the dynamic type test
void doDynamicTypeTest()
{
   int            reps = 500;
   unsigned int   objects = 500000;
   unsigned int   total_time = 0;
```

```
   cout << "Performing dynamic type test for "<< objects << " objects." <<
endl << endl;
   cout << "This test is designed to determine how efficiently the language
allows " << endl;
   cout << "a program to determine if an object is an instance of a
particular class" << endl;
   cout << "at run-time.   This type of testing is done in C++ via the
dynamic_cast<> " << endl;
   cout << "operator which requires that the code be built with RTTI enabled.
The " << endl;
   cout << "test will create an instance of a derived class object and store
it in a" << endl;
   cout << "base class pointer.  It will then perform the dynamic cast to
test if the" << endl;
   cout << "object is a derived class object.  If it is, the test will call a
method" << endl;
   cout << "which is defined in the derived class only.  If it is not an
error message" << endl;
   cout << "will be printed." << endl << endl;

   for(int x = 0; x < reps; x++)
   {
      total_time += dynamicTypeTest(objects);
   }
   cout << "Average time for dynamic type test is "<< total_time/reps << "
milliseconds for " << objects << " objects." << endl << endl << endl;
}


/*****************************************************************
   The purpose of this test is to determine the efficiency with which
   the language determines if an object is of a particular type.
*****************************************************************/
unsigned int dynamicTypeTest(unsigned int reps)
{
   clock_t start_clock = clock();

   BaseObject *pbo = new DerivedObject();
   DerivedObject *pdo = NULL;

   for(unsigned int x = 0; x < reps; x++)
   {
      pdo = dynamic_cast<DerivedObject*>(pbo);
      if(pdo != NULL)
      {
         pdo->derivedOnlyCall();
      }
      else
      {
         cout << "Test error: dynamic cast failure." << endl;
      }

   }
   delete pbo;

   clock_t end_clock = clock();
```

```
   return clocks_to_millis(start_clock, end_clock);
}


//driver for the floating point test
void doFloatingPointTest()
{
   int            reps = 500;
   unsigned int   elements = 100000;
   unsigned int   total_time = 0;

   srand( (unsigned)time( NULL ) );

   cout << "Performing floating point test for "<< elements << " elements."
<< endl << endl;
   cout << "This test will create an array of the specified number of
elements " << endl;
   cout << "it will then iterate through all elements in the array four
times." << endl;
   cout << "The first time it will copy a double out of the array, multiply
it" << endl;
   cout << "by itself and copy it back into the array.  The second pass will
copy" << endl;
   cout << "the element out of the array, divide it by 3.14 then copy it back
into the array." << endl;
   cout << "The third pass will copy the element out of the array, add it to
itself" << endl;
   cout << "and place it back in the array.  The final pass will copy each
element" << endl;
   cout << "subtract 3.14 from it and place it back in the array." << endl <<
endl;

   for(int x = 0; x < reps; x++)
   {
      total_time += floatingPointTest(elements);
   }
   cout << "Average time for floating point test is "<< total_time/reps << "
milliseconds for " << reps << " repetitions." << endl << endl << endl;
}


/*****************************************************************
   This test is designed to test the speed of array manipulation
   and basic mathematical functions for floating point types.  While
   clocking it will create an array of the specified number of doubles
   and walk it four times.  Each time it will pull out an element and
   perform a mathematical operation, then it will replace the element
   in the array.
*****************************************************************/
unsigned int floatingPointTest(unsigned int elements)
{
   double *mydoubles = new double[elements];
   double temp = 0.0;

   for(unsigned int x = 0; x < elements; x++)
   {
      mydoubles[x] = 1.7567;
```

```
   }

   clock_t start_clock = clock();

   for(x = 0; x < elements; x++)
   {
      temp = mydoubles[x];
      temp *= temp;
      mydoubles[x] = temp;
   }

   for(x = 0; x < elements; x++)
   {
      temp = mydoubles[x];
      temp /= 3.14;
      mydoubles[x] = temp;
   }

   for(x = 0; x < elements; x++)
   {
      temp = mydoubles[x];
      temp += temp;
      mydoubles[x] = temp;
   }

   for(x = 0; x < elements; x++)
   {
      temp = mydoubles[x];
      temp -= 3.14;
      mydoubles[x] = temp;
   }

   clock_t end_clock = clock();

   delete[] mydoubles;

   return clocks_to_millis(start_clock, end_clock);
}


//driver for the string test
void doStringTest()
{
   int            reps = 500;
   unsigned int   strings = 10000;
   unsigned int   total_time = 0;


   cout << "Performing string manipulation test for "<< strings << "
strings." << endl << endl;
   cout << "This test will copy a string into a temporary string.  It will
then " << endl;
   cout << "compare the two strings using a string sensitive comparison.
Next it " << endl;
   cout << "will convert the temporary string to upper case.  It will then
compare " << endl;
   cout << "strings again." << endl << endl;
```

```
   for(int x = 0; x < reps; x++)
   {
      total_time += stringTest(strings);
   }
   cout << "Average time for string manipulation test is "<< total_time/reps
<< " milliseconds for " << reps << " repetitions." << endl << endl << endl;
}


/*****************************************************************
   This test is designed to test the efficiency of string manipulations
   and comparisons.
*****************************************************************/
unsigned int stringTest(unsigned int reps)
{
   unsigned int x = 0;
   CString init_string("Initial string for manipulation test");
   CString cmp_string("Initial string for manipulation test");
   CString tmp_string = "";
   CString tmp_string2 = "";

   clock_t start_clock = clock();

   for(x = 0; x < reps; x++)
   {
      tmp_string = init_string;
      if(!tmp_string.Compare(cmp_string))
      {
         tmp_string.MakeUpper();
         tmp_string2 = tmp_string;
         if(!tmp_string2.Compare(cmp_string))
         {
            cout << "Test error strings should not be equal anymore..." <<
endl;
         }
      }
      else
      {
         cout << "Test error strings should be equal..." << endl;
      }
   }

   clock_t end_clock = clock();

   return clocks_to_millis(start_clock, end_clock);
}
```

B.2 OBJECTS.H

```
#ifndef _OBJECTS_H_
#define _OBJECTS_H_


class MutableObj
{
   public:
      MutableObj():m_counter(0){};
      static void MutateObject(MutableObj &mo){mo.Mutate();}
      inline void Mutate(){m_counter++;}

   private:
      int m_counter;
};

class BaseObject
{
   public:
      BaseObject():m_counter(0){};
      virtual ~BaseObject(){};
      virtual void polymorphicCall();

   protected:
      int m_counter;
};

class DerivedObject : public BaseObject
{
   public:
      DerivedObject(){};
      virtual ~DerivedObject(){};
      virtual void polymorphicCall();
      virtual void derivedOnlyCall() inline;
};

#endif
```

Appendix C - Java Source Code

C.1 SPEED.JAVA

```
import java.lang.*;

public final class Speed
{
   //create a Speed object and invoke the test methods.
   public static void main(String[] args)
   {
      System.out.println("Java Language Performance Test Results");
      System.out.println("***************************************");
      System.out.println("");

      Speed s = new Speed();
      s.doIntArrayTest();
      s.doFloatingPointTest();
      s.doStringTest();
      s.doMutableObjectTest();
      s.doPolymorphicTest();
      s.doDynamicTypeTest();
   }


   //constructor for a speed method
   public Speed()
   {
   }


   //driver for the integer array and math test
   public void doIntArrayTest()
   {
      int reps = 500;
      int elems = 100000;
      long total_time = 0;

      System.out.println("Performing int array test for " +   elems + "
elements.\n");
      System.out.println("This test will create an array of the specified
number of elements");
      System.out.println("it will then iterate through all elements in the
array four times.");
      System.out.println("The first time it will copy an integer out of the
array, increment it");
      System.out.println("and copy it back into the array.  The second pass
will copy the element");
      System.out.println("out of the array, decrement it by one and copy it
back into the array.");
      System.out.println("The third pass will copy the element out of the
array, multiply it");
      System.out.println("and place it back in the array.  The final pass
will copy each element");
```

```
      System.out.println("divide it by two, and place it back in the
array.");

      for(int x= 0; x < reps; x++)
      {
         total_time += intArrayTest(elems);
      }
      System.out.println("");
      System.out.println("Average time for int array test is " +
total_time/reps + " milliseconds for "+ reps + " repetitions.");

      System.out.println("");
      System.out.println("");
   }


   /*****************************************************************
      This test is designed to test the speed of array manipulation
      and basic mathematical functions for basic types.  While clocking
      it will create an array of the specified number of ints and walk
      it four times.  Each time it will pull out an element and perform
      a basic mathematical operation, then it will replace the element
      in the array.
   *****************************************************************/

   public long intArrayTest(int len)
   {
      int test = 0;
      int x = 0;

      //start the timer
      long start_time = System.currentTimeMillis();

      int[] myints = new int[len];
      for(x= 0; x < len; x++)
      {
         test = myints[x];
         test++;
         myints[x] = test;
      }
      for(x= 0; x < len; x++)
      {
         test = myints[x];
         test--;
         myints[x] = test;
      }
      for(x= 0; x < len; x++)
      {
         test = myints[x];
         test*=2;
         myints[x] = test;
      }
      for(x= 0; x < len; x++)
      {
         test = myints[x];
         test/=2;
         myints[x] = test;
```

```
      }

      long end_time = System.currentTimeMillis();
      long time_required = end_time - start_time;

      return time_required;
   }


   //driver for the mutable object test
   public void doMutableObjectTest()
   {
      int   reps = 500;
      int   calls = 500000;
      long  total_time = 0;

      System.out.println("Performing mutable object test for " + calls + "
method calls.");
      System.out.println("This test will create an object which has a single
integer member");
      System.out.println("variable and a public method which, when called,
increments this");
      System.out.println("member.  The class of this object will also have a
static method");
      System.out.println("which takes a mutable object as its lone parameter.
This allows us");
      System.out.println("to test the efficiency with which the language
passes parameters by");
      System.out.println("reference.  The test creates a mutable object and
passes it to the");
      System.out.println("class method which then invokes the mutator method
on the passed object.");

      total_time = 0;
      for(int x= 0; x < reps; x++)
      {
         total_time += mutableObjectTest(calls);
      }
      System.out.println("");
      System.out.println("Average time for mutable object test is " +
total_time/reps + " milliseconds for "+ reps + " repetitions.");
      System.out.println("");
      System.out.println("");
   }


   /*****************************************************************
      This test is designed to test static method calls and simple
      object state change calls.  While clocking it will create a
      new object and call a static method which will call a mutator
      method on the object for the specified repetitions.
   *****************************************************************/
   public long mutableObjectTest(int reps)
   {
      long start_time = System.currentTimeMillis();

      MutableObj mo = new MutableObj();
```

```
      for(int x = 0; x < reps; x++)
      {
         MutableObj.mutateObject(mo);
      }

      long end_time = System.currentTimeMillis();

      return end_time - start_time;
   }


   //driver for the polymorphic call test
   public void doPolymorphicTest()
   {
      int   reps = 500;
      int   calls = 500000;
      long  total_time = 0;

      System.out.println("Performing polymorphic test for " + calls + "
method calls.");
      System.out.println("This test is designed to test the efficiency with
which the language");
      System.out.println("handles method invocations which require dynamic
binding.  The test");
      System.out.println("creates a derived class object and assigns it to a
base class pointer.");
      System.out.println("It then invokes a virtual method on the base class
object which will ");
      System.out.println("be handled by the overriding derived class method.
Dynamic binding is");
      System.out.println("crucial to engineering reusable object oriented
components.");

      for(int x= 0; x < reps; x++)
      {
         total_time += polymorphicTest(calls);
      }
      System.out.println("");
      System.out.println("Average time for polymporphic test is " +
total_time/reps + " milliseconds for "+ reps + " repetitions.");
      System.out.println("");
      System.out.println("");
   }


   /*********************************************************************
      The purpose of this test is to determine the efficiency with which
      the language processes method calls which require dynamic binding.
   *********************************************************************/
   public long polymorphicTest(int reps)
   {
      long start_time = System.currentTimeMillis();

      BaseObject obj = new DerivedObject();
      for(int x = 0; x < reps; x++)
      {
         obj.polymorphicCall();
```

```
      }

      long end_time = System.currentTimeMillis();

      return end_time - start_time;
   }


   //driver for the dynamic type test
   public void doDynamicTypeTest()
   {
      int   reps = 500;
      int   objects = 500000;
      long  total_time = 0;

      System.out.println("Performing dynamic type test for " + objects + "
objects.");

      System.out.println("This test is designed to determine how efficiently
the language allows ");
      System.out.println("a program to determine if an object is an instance
of a particular class");
      System.out.println("at run-time.  This type of testing is done in java
via the instanceof ");
      System.out.println("operator which is a built-in language feature.  The
test will create an");
      System.out.println("instance of a derived class object and store it in
a base class pointer.");
      System.out.println("It will then perform the dynamic cast to test if
the object is a derived");
      System.out.println("class object.   If it is, the test will call a
method which is defined in the ");
      System.out.println("derived class only.  If it is not an error message
will be printed");

      for(int x = 0; x < reps; x++)
      {
         total_time += dynamicTypeTest(objects);
      }
      System.out.println("");
      System.out.println("Average  time  for  dynamic  type  test  is  "  +
total_time/reps + " milliseconds for "+ reps + " repetitions.");
      System.out.println("");
      System.out.println("");
   }


   /*********************************************************************
      The purpose of this test is to determine the efficiency with which
      the language determines if an object is of a particular type.
   *********************************************************************/
   public long dynamicTypeTest(int reps)
   {
      long start_time = System.currentTimeMillis();

      BaseObject obj = new DerivedObject();
      for(int x = 0; x < reps; x++)
```

```
      {
         if(obj instanceof DerivedObject)
         {
            ((DerivedObject)obj).derivedOnlyMethod();
         }
         else
         {
            System.out.println("Error: dynamic type test failed.");
         }
      }

      long end_time = System.currentTimeMillis();

      return end_time - start_time;
   }


   //driver for the floating point math test
   public void doFloatingPointTest()
   {
      int reps = 500;
      int elements = 100000;
      long total_time = 0;

      System.out.println("Performing floating point test for " + elements + "
elements.");
      System.out.println("This test will create an array of the specified
number of elements");
      System.out.println("it will then iterate through all elements in the
array four times.");
      System.out.println("The first time it will copy a double out of the
array, multiply it");
      System.out.println("by itself and copy it back into the array.  The
second pass will copy");
      System.out.println("the element out of the array, divide it by 3.14
then copy it back into the array.");
      System.out.println("The third pass will copy the element out of the
array, add it to itself");
      System.out.println("and place it back in the array.  The final pass
will copy each element");
      System.out.println("subtract 3.14 from it and place it back in the
array.");

      for(int x= 0; x < reps; x++)
      {
         total_time += floatingPointTest(elements);
      }
      System.out.println("");
      System.out.println("Average time for floating point test is " +
total_time/reps + " milliseconds for "+ reps + " repetitions.");

      System.out.println("");
      System.out.println("");
   }


   /*****************************************************************
```

```
      This test is designed to test the speed of array manipulation
      and basic mathematical functions for floating point types.  While
      clocking it will create an array of the specified number of doubles
      and walk it four times.  Each time it will pull out an element and
perform
      a mathematical operation, then it will replace the element
      in the array.
   ***************************************************************/
   public long floatingPointTest(int elements)
   {
      double temp = 0;
      int x = 0;

      //build an array of doubles
      double[] mydoubles = new double[elements];
      for(x=0; x < elements; x++)
      {
         mydoubles[x] = 1.7567;
      }

      //start the timer
      long start_time = System.currentTimeMillis();

      for(x = 0; x < elements; x++)
      {
         temp = mydoubles[x];
         temp *= temp;
         mydoubles[x] = temp;
      }

      for(x = 0; x < elements; x++)
      {
         temp = mydoubles[x];
         temp /= 3.14;
         mydoubles[x] = temp;
      }

      for(x = 0; x < elements; x++)
      {
         temp = mydoubles[x];
         temp += temp;
         mydoubles[x] = temp;
      }

      for(x = 0; x < elements; x++)
      {
         temp = mydoubles[x];
         temp -= 3.14;
         mydoubles[x] = temp;
      }

      long end_time = System.currentTimeMillis();
      long time_required = end_time - start_time;

      return time_required;
   }
```

```
   //driver for the string manipulation test
   public void doStringTest()
   {
      int   reps = 500;
      int   strings = 10000;
      long  total_time = 0;

      System.out.println("Performing string manipulation test for " + strings
+ " strings.");
      System.out.println("This  test  will  copy  a  string  into  a  temporary
string.  It will then ");
      System.out.println("compare  the  two  strings  using  a  string  sensitive
comparison.  Next it ");
      System.out.println("will  convert  the  temporary  string  to  upper  case.
It will then compare ");
      System.out.println("strings again.");

      for(int x = 0; x < reps; x++)
      {
         total_time += stringTest(strings);
      }
      System.out.println("");
      System.out.println("Average  time  for  string  manipulation  test  is  " +
total_time/reps + " milliseconds for "+ reps + " repetitions.");
      System.out.println("");
      System.out.println("");
   }


   /**********************************************************************
      This test is designed to test the efficiency of string manipulations
      and comparisons.
   **********************************************************************/
   public long stringTest(int reps)
   {
      int x = 0;
      final String init_string = "Initial string for manipulation test";
      final String cmp_string = "Initial string for manipulation test";
      String tmp_string = "";
      String tmp_string2 = "";

      long start_time = System.currentTimeMillis();

      for(x = 0; x < reps; x++)
      {
         tmp_string = init_string;
         if(tmp_string.equals(cmp_string))
         {
            tmp_string2 = tmp_string.toUpperCase();
            if(tmp_string2.equals(cmp_string))
            {
               System.out.println("Test  error  strings  should  not  be  equal
anymore...");
            }
         }
         else
```

```
            {
               System.out.println("Test error strings should be equal...");
            }
         }

         long end_time = System.currentTimeMillis();

         return end_time - start_time;
      }
}
```

C.2 BASEOBJECT.JAVA

```
public class BaseObject
{
   public BaseObject()
   {
      m_counter = 0;
   }

   public void polymorphicCall()
   {
      System.out.println("Error call went to base class.");
   }

   protected int m_counter;
}
```

C.3 DERIVEDOBJECT.JAVA

```
public final class DerivedObject extends BaseObject
{
   public DerivedObject(){};
   public void polymorphicCall()
   {
      m_counter++;
   }

   public void derivedOnlyMethod()
   {
      m_counter++;
   }
}
```

C.4 MUTABLEOBJ.JAVA

```
public final class MutableObj
{
   public MutableObj(){m_counter = 0;}
   public static void mutateObject(MutableObj mo){mo.mutate();}
   public void mutate(){m_counter++;}
   private int m_counter;
}
```